Feature-based Elicitation of Cognitively Efficient Visualizations for SPL Configurations

Sauvage-Thomase, Céline; Biri, Nicolas; Perrouin, Gilles; Genon, Nicolas; Heymans, Patrick

Published in:
Human Centered Software Product Lines

Publication date:
2017

Document Version
Peer reviewed version

Feature-based Elicitation of Cognitively Efficient Visualizations for SPL Configurations

Céline Sauvage-Thomase, Nicolas Biri, Gilles Perrouin, Nicolas Genon, and Patrick Heymans

Abstract Configuring a SPL is a cognitively difficult activity that requires a deep understanding of the features and their constraints to be performed effectively. To this end, SPL configurators have been equipped with various visualizations to assist users in their tasks. However, there are many ways to visualize data: the process of associating an efficient visualization to a given (configuration) task is neither well-understood nor systematically applied, resulting in confusing visualizations yielding configuration errors. In this chapter, we offer such a process, based on theories of the visualization community for data representation. The first step consists in choosing the data to be visualized. This selection induces restrictions on the types of visualization that are then computed based on the data characteristics and best practices from semiology and visual languages. Designers can then select an efficient visualization for the intended task. Our process is supported by feature models and FAMILIAR to merge and constrain the set of applicable visualizations.

Celine Thomase-Sauvage
Luxembourg Institute of Technology, Luxembourg e-mail: celine.thomase@list.lu

Nicolas Biri
Luxembourg Institute of Technology, Luxembourg e-mail: nicolas.biri@list.lu

Gilles Perrouin
PReCISE, Faculty of Computer Science, University of Namur, Belgium
e-mail: gilles.perrouin@unamur.be

Nicolas Genon
PReCISE, Faculty of Computer Science, University of Namur, Belgium
e-mail: nicolas.genon@unamur.be

Patrick Heymans
PReCISE, Faculty of Computer Science, University of Namur, Belgium
e-mail: patrick.heymans@unamur.be

1 Introduction

The activity of configuring a product in the context of a Software Product Line (SPL) is cognitively challenging. Indeed, the engineer has for instance to understand the complex relationships of the features involved in the configuration of his product. Even if solving these dependencies can be automated (e.g. by using a SAT solver in the back-end), it does not help regarding configuration understanding. To this end, current configurations tools integrate some visualization supports intended to help users in their tasks. However there is more than a single way to visually represent a configuration [16] and we argue in this paper that the process of associating a visualization to a given task is not well-understood and mostly results in generic visualizations. These visualizations are clearly sub-optimal to perform some tasks such as the understanding of propagations during configuration.

To provide dedicated visualizations, we present a method to explicitly model and guide visualization choices in terms of feature models (FM) and relate them with the kind of data involved for a particular task. The kinds of data involved in the configuration task (e.g. features, constraints) are defined on a first FM named dataset FM. The possible visualization designs (e.g. trees, maps, etc.) are also represented with FMs, one by visualization design, named visualization designs FMs. The dataset FM configuration induces restrictions on the possible configurations of the visualization design FMs thanks to a set of mapping rules. These rules are based on the types of the configured data and their possible representations by the visual properties [2] of the visualization designs where a visual property is a graphical element that can be perceived by the human eye (e.g color, shape, size). Each visualization design FM offers different choices for visual properties assignment. To ease the configuration task, these visualization FMs are merged using the FAMILIAR environment [1] in a single visualization FM and configuration constraints are automatically generated. Hence, the valid configurations of the visualization FM forms thus the set of appropriate visualizations for the considered input data.

The remainder of this chapter is organized as follows: Section 2 illustrates our approach on a configuration task issue example. Section 3 describes the different steps of our method. Section 4 applies the method through our running example. Section 5 provides an overview of the visualization designs guidance approaches existing in the literature. Finally, Section 6 concludes this paper by wrapping up our contributions and highlighting some future perspectives.

2 Example

To illustrate the challenge of visualization choice, we focus on issues occurring during product configuration in SPL. In an industrial environment, feature models tend to be complex, involving an important number (thousands) of features and complex crosstree constraints [20, 18].

In [4], Deelstra et al. pointed out that in large feature models involving a lot of complex cross-tree constraints, engineers can not accurately see how their choices impact other features. This difficulty is twofold: on the one hand, numerous cross-tree constraints make the consequences of a choice difficult to forecast, on the other hand, a large number of features can make the configuration impact difficult to spot. Our task is thus to select an adapted visualization to ease the comprehension of a feature selection during the configuration of the FM.

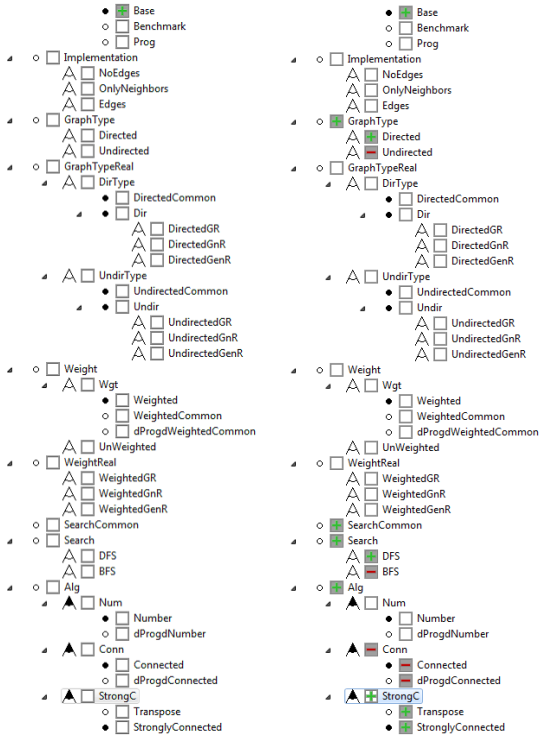

Fig. 1 Feature IDE configuration views

We take the Graph Product Line (GPL) [7] as a running example of product line configuration throughout this chapter. On the GPL feature model, we start a configuration with the selection of the following features `Gpl, MainGpl, Test, StartHere and Base`. From this point, we continue with the selection of the feature `StrongC`. In Figure 1, the screenshots on the left hand side and on the right hand side display respectively the configuration *before* and *after* the feature selection in the Feature IDE configuration tool. Similarly, in Figure 2, the screenshots on top and on bottom show respectively the state of the S2T2 configuration tools *before* and *after* the selection. The distinction of the feature names in Figures 1 and 2 is not important. The focus is on the fact that a feature is selected/rejected.

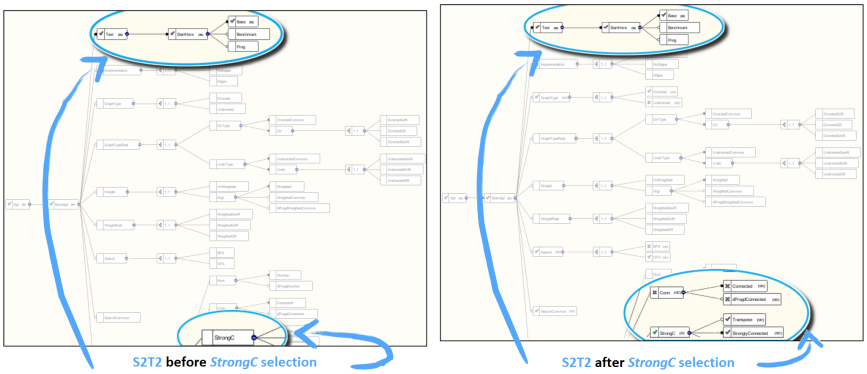

Fig. 2 S2T2 configuration views

First, we must acknowledge that none of the proposed tools explicitly claims to tackle the choice understanding issue. Nevertheless, none of the two editors makes a visual difference between the features that are selected or rejected *before* the `StrongC` selection and those becoming selected or rejected *after* the selection. As a consequence, it is difficult to figure out the differences *before* and *after* the selection. S2T2 differentiates the user choices from constraint propagation, which can partially support the user, but it does not directly address the issue.

In Section 4, we will apply our visualization selection method to this example and show how we can explicitly represent the choice consequences in a both immediate and meaningful way.

3 Visualization elicitation method

We propose an SPL-backed method that guides the selection of an appropriate visualization for the data involved in a configuration user task. Though the models offered are dedicated to the support SPL configuration tasks, the method can be adapted to any data visualization needs.

The method follows a traditional two-fold SPL approach [13]. First, the domain engineering phase, led by a *visualization domain engineer*, consists in the definition of the FMs that will support the selection of the visualization. Second, the application engineering phase, led by the *configuration visualization engineer* that will configure these models and fine tune the selected visualization that will be provided to the *end-user*. The process is illustrated in Figure 3.

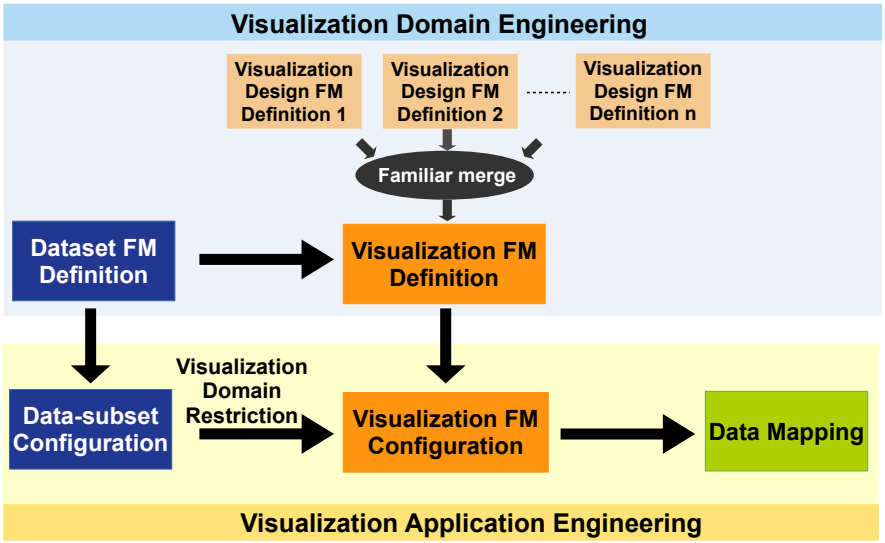

Fig. 3 Visualization engineering process

3.1 Visualization domain engineering

The visualization domain engineering phase consists in the formalization of two feature models. A *dataset FM*, specific to an application domain, which allows the configuration of the dataset to visualize, and the *visualization FM*, a variability model that represents all the visualization possibilities that the visualization domain engineer will offer to the configuration engineer. This second model is data agnostic and thus can be reused from one domain to the other.

3.1.1 Dataset FM

During this phase, the visualization domain engineer defines a FM that represents the datasets that can potentially be interesting to provide insight to the end user.

The definition of the dataset FM is built on the multidimensional concepts formalised in the context of OnLine Analytical Processing (OLAP) research [14]. These concepts contain the notions of *dimensions* and *measures*. A *dimension* is somehow a concern on the data. More precisely it defines an analysis axis formed by a set of data with the same datatype, providing a base on which the other data are analyzed. A *measure* describes data that have to be analyzed over the dimensions. Bearing in mind these definitions, the dataset FM is set with three subtrees (see Figure 4). One for the set of *dimensions* involved in a configuration process, another for the set of *measures* associated with these dimensions and the last for the set of

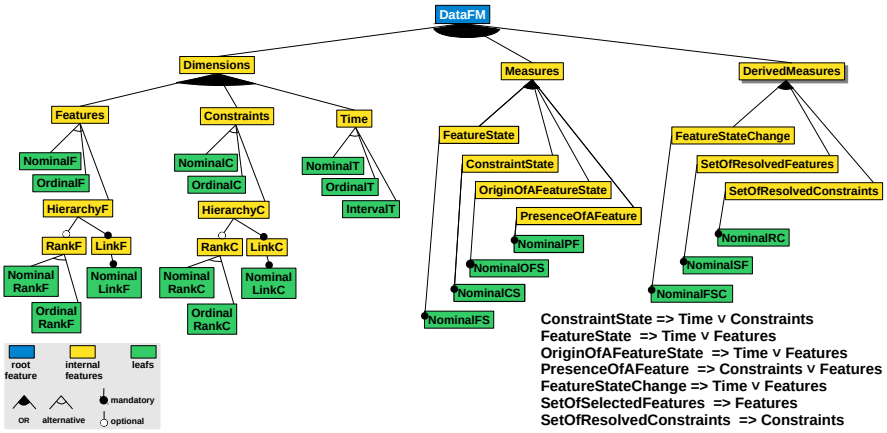

Fig. 4 Dataset FM

derived measures deduced from these *measures*.

To ease the mapping with visual solutions, we also need to precise the datatypes associated to the data constituting the dimensions and measures. To do so, we use the four types of measurement scale cited by Stevens in [21], which provide precise types of scale for the dimensions and the measures [24]. The first type is *nominal* and defines the possibility for elements to be distinguishable. The second type is *ordinal* and means that the elements can be sorted depending on a rank order. The third type is named *interval* and adds the capacity to calculate the difference between two elements. Finally, the type *ratio* is the more expressive. It defines the additional possibility to calculate a ratio between two elements. These datatypes are defined on the leaf features of the three subtrees *dimensions*, *measures* and *derived measures*. It has to be noticed that several types can be associated with the same feature to let the possibility to represent a *dimension*, a *measure* or a *derived measure* with a less expressive datatype than initially needed. The prefixes `Interval`, `Ordinal` and `Nominal` of the leaf features will be used as types to restrict the possibilities in the visualization models during the application engineering phase (see section 3.2.3).

In the following paragraphs, we will detail the set of *dimensions*, *measures* and *derived measures* with their associated datatypes that are involved in a configuration process.

Dimensions

Configuring a feature model means making successive choices on features in accordance with the constraints expressed on the feature model. From this definition,

three dimensions can be deduced and expressed on the data feature model by the visualization domain engineer.

Features dimension. Given that a configuration is carried out by selecting features, the set of features forms an obvious dimension which is identified as `Features` in Figure 4. The dimension `Features` has three children in the dataFM: `NominalF`, `OrdinalF` and `HierarchyF`. The `HierarchyF` child is defined to reflect the hierarchical organization of the features on a feature model. Actually, the notion of hierarchy encompasses a mandatory parent-child relationship (`LinkF` in the dataset FM) and an optional rank of the elements inside the hierarchy (`RankF` in the dataset FM), this decomposition is motivated by the visual representation of hierarchy detailed in Section 3.1.2. The parent-child relationship has a nominal datatype (`NominalLinkF` in the dataset FM) because the information to convey is the existence of a relationship between the linked elements. The rank requires an ordinal datatype (`NominalRankF` in the dataset FM) as there is a specific order between the ranks of the hierarchy (i.e., rank 1 ¡ rank 2 ¡ ...). However, a representation of this hierarchical organization is not always necessary. The children `NominalF` and `OrdinalF` define the datatypes nominal and ordinal and allow to represent only the feature names or respectively an ordered list of the feature names.

Constraints dimension. The feature choices are restricted by the feature model constraints. Thus, another relevant dimension is the `Constraints`. The `constraints` dimension is composed by two types of constraints. The first type is formed by the hierarchical constraints between the child nodes and the parent nodes of the feature tree. The second type is the cross-tree constraints scattered over the whole feature model. All the constraints can be represented with an algebraic tree where the internal nodes are logical operators and the leafs are logical literals involving a feature. Resultantly the `constraints` dimension has also a hierarchical organization and its datatypes are defined in the same way as those of the `constraints` dimension.

Time dimension. During a configuration process successive choices are made. The `Time` dimension allows the analysis of the configuration process at different points in time. We consider that the configuration process is constituted by a set of configuration steps where each step contains a user choice which may be followed by an automatic propagation of constraints. The `Time` dimension is formed with this set of configuration steps. The relevant associated datatype for the time dimension is interval, named as `IntervalT` in the dataset FM. However if the time difference between two time data does not need to be represented, the datatype ordinal is adequate and the feature `ordinalT` is selected. Similarly, if the representation of the order between two time data is not useful, the feature `NominalT` representing the datatype nominal is chosen.

Measures

The visualization domain engineer defines on the dataset FM the set of measures that are involved during a configuration process. As previously mentioned, the measures

are associated with one or more dimensions. To visualize a measure, at least one of its dimensions must also be present in the dataset. Consequently, to ensure the data feature model consistency, cross-tree constraints are needed expressing that the measures can not exist without at least one of their dimensions. Hence, for each measure added in the data feature model, a cross-tree constraint is created involving the measure and its associated dimensions.

Feature state. This measure is straightforward as it denotes the state of a feature during a configuration. A feature is selected, rejected or choice free. Its associated dimensions are the `Time` and the `Features`. Its datatype is nominal. In Figure 4 this measure is identified as `FeatureState` and its datatype as `NominalFS`.

Constraint state. This measure identifies the state of a constraint which can be resolved or not resolved during a configuration process. Its associated dimensions are `Time` and `Constraints`. Its datatype is nominal. This measure is named as `ConstraintState`, its datatype as `NominalCS`.

Feature state origin. This measure distinguish whether a feature is selected or rejected following a user action (chosen) or following an automatic propagation of constraints (deduced). Its associated dimensions are `Time` and `Features`. Its datatype is nominal. This measure is named as `OriginOfAFeatureState` and its datatype as `NominalOFS`.

Feature-constraint involvement. This measure determines if a feature is part of a given constraint. Its associated dimensions are `Constraints` and `Features`. Its datatype is nominal. This measure is identified as `PresenceOfAFeature` and its datatype as `NominalPF`.

The derived measures

These measures are deduced from the main measures that we have just described. It has to be noticed that this list of deduced measures is not exhaustive.

Feature state change. This measure allows to know whether a feature becomes selected or rejected between two given configuration steps. It is deduced from the measure `FeatureState` by identifying the feature states which are modified between the two given configuration steps. Thus, it is associated with the same dimensions: `Time` and `Features`. Its datatype is nominal. In Figure 4, this measure is named as `FeatureStateChange` and its datatype as `NominalFSC`.

Set of selected features. This measure shows the set of selected features at a given configuration step. It is deduced from the measure `FeatureState`. Resultantly, it is associated with `Time` and `Features`. Its datatype is nominal. This measure is named as `SetOfSelectedFeatures` and its datatype as `NominalSF`. According to the measure semantics, this measure is irrelevant without the selection of the `Features` dimension. Consequently, while for the other measures one of the dimensions associated with is indifferently mandatory, for this measure the `Features` dimension is specifically mandatory. The following constraint is added to the feature model:
`SetOfSelectedFeatures` $\Rightarrow$ `Features`.

Set of resolved constraints. This measure defines the set of resolved constraints at a given configuration step. It is deduced from the measure `ConstraintState` Thus, it is associated with with `Time` and `Constraints`. Its datatype is nominal. This measure is named as `SetOfSelectedConstraints` and its datatype as `NominalSC`. Similarly to the measure `SetOfSelectedFeatures`, the following constraint is added to the feature model:
`SetOfResolvedConstraints` $\Rightarrow$ `Constraints`.

3.1.2 Visualization FM

The second task of the visualization domain engineer is to characterize the visualizations in the form of a FM called the visualization FM. In the following, a visualization refers to a complete visualization design. A treemap [6] illustrated in Figure 5 or a sunburst [19] illustrated in Figure 6 are some examples of visualization designs. A visualization design is defined in accordance with a set of visual properties and a series of constraints. The visualization FM is defined in several steps. First, we will define a FM that characterizes the set of visual variables and their capacity expression in terms of data. Then, on this base, the visualization domain engineer provides, for each proposed visualization design, a visualization design FM allowing the configuration of the visual properties. Finally, the set of defined visualization design FMs are merged to obtain the visualization FM unifying the set of valid configurations of each visualization design FM.

Fig. 5 Treemap designed with Calluna, a LIST visualization solution

FM for visual variables. We start with the definition of a reference FM to describe the levels of measurement of each of the visual properties. The level of mea-

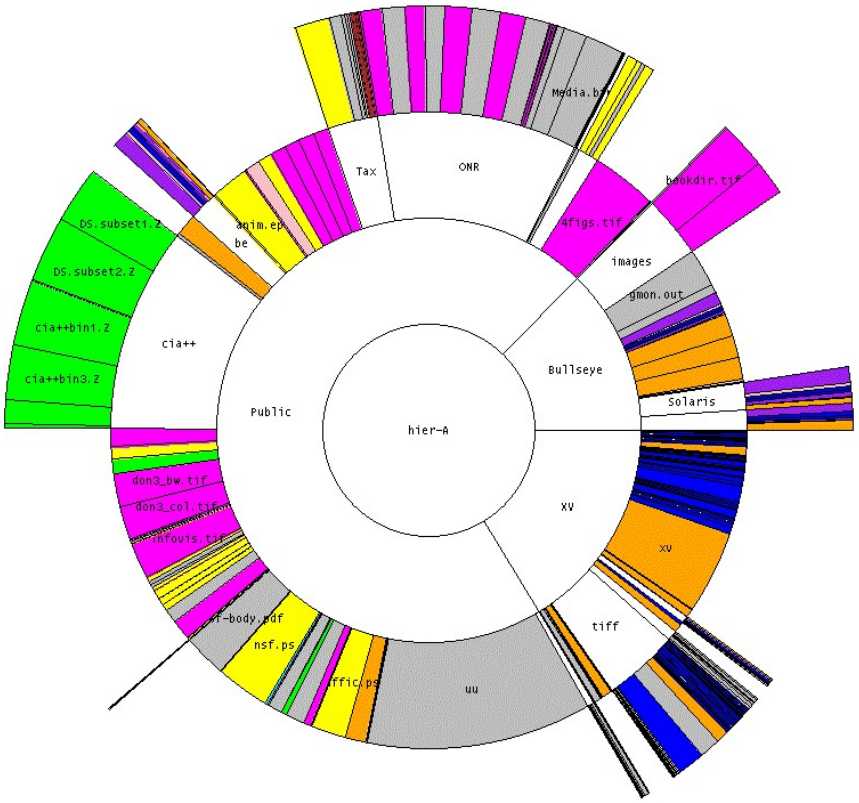

Fig. 6 Sunburst (depiction of a file hierarchy [19])

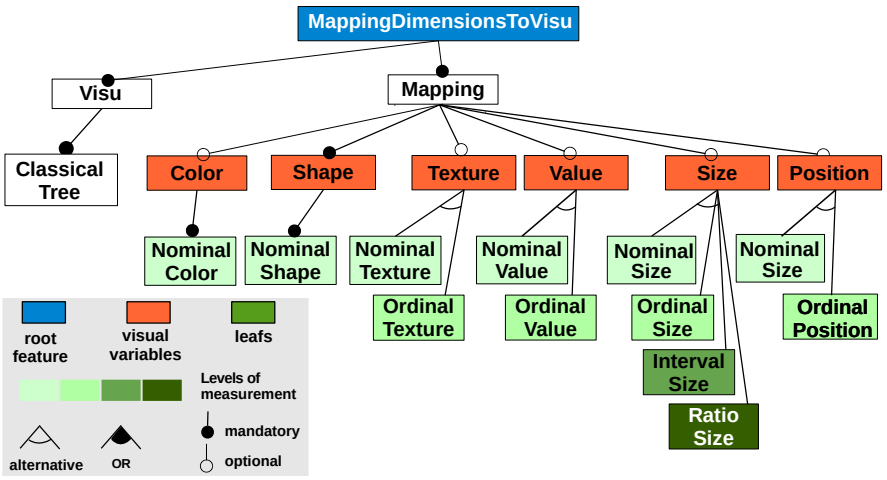

Fig. 7 Classical Tree FM

surement denotes the highest datatype that the property can express. There exists several classifications of the visual properties according to their levels of measurements [2, 3, 9, 8, 23]. We settle on Bertin's proposal [2]. At the current stage, we do not claim the FM to be complete, and we define it in a scalable way to support further extensions of the mapping rules.

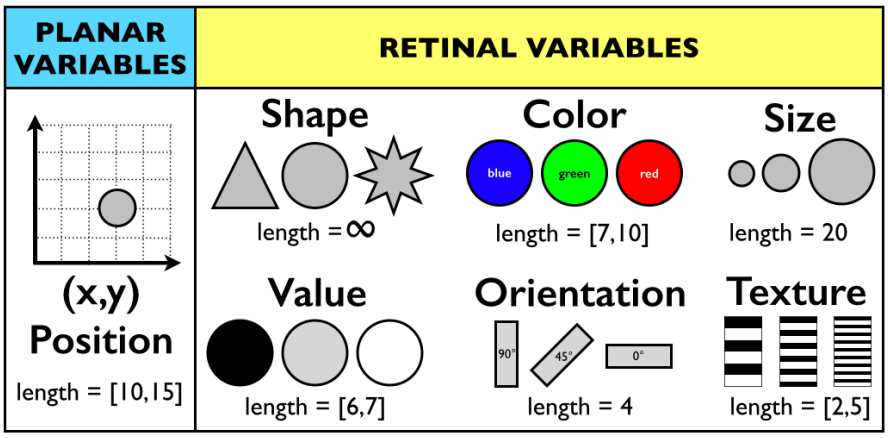

Fig. 8 The 8 visual variables defined by Bertin [2]

Bertin defines 8 visual variables that are the building blocs to design any symbol (see Figure 8). The variables spread out into 2 categories: the planar variables (x and y position on a 2D plan) and six retinal variables (shape, color, orientation, size, texture and value (aka., brightness)). Each visual variable is suitable to express a certain level of measurement. More precisely, this referent FM maps the dimensions and measures datatypes onto the visual variables' levels of measurement. As represented by Figure 9, the root of this FM is the `Mapping` feature. This feature has seven children that denote each of the visual variables (except for the planar variables that are gathered under the feature `Position`). The `Shape`, `Orientation` and `Color` have respectively one child that indicates a nominal level of measurement (i.e., the features `NominalShape`, `NominalOrientation` and `NominalColor`). `Texture` and `Value` are ordinal variables and hence have respectively two children: `NominalTexture`, `OrdinalTexture` and `NominalValue`, `OrdinalValue`. The variables of `Size` and (x,y) `Position` have the highest level of measurement: each of them have four children that correspond to the nominal, ordinal, interval and ratio levels of measurement.

The hierarchy notion elicited in Section 3.1.1 has no immediate correspondence to the levels of measurement of the visual variables. However, as the hierarchy is expressed in the dataset FM with a mandatory nominal datatype and an ordinal datatype, it can be mapped to a pair (nominal, ordinal) of visual variables' levels of measurement or to a simple nominal level. For example, in the usual representation

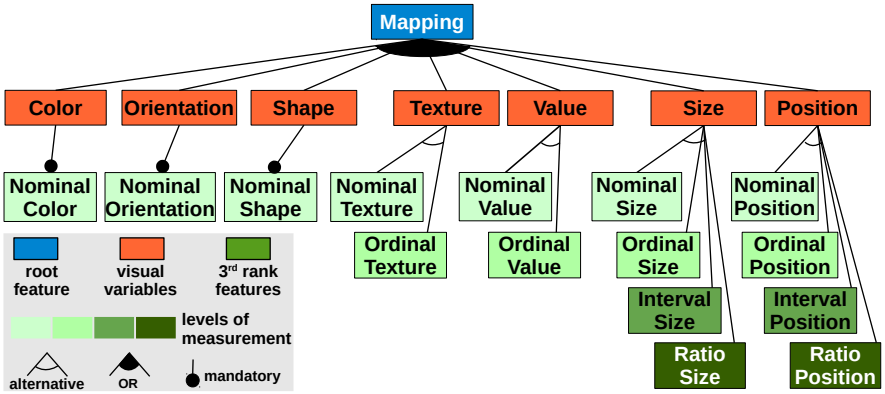

Fig. 9 Mapping between the visual variables and their levels of measurement

of FM, the hierarchy is expressed by the *shape* and *y position* visual variables: parent elements are linked to each child element by a line that is depicted with some *shape*. Moreover, the parent elements are located on the top of the diagram, while their children lie below, which corresponds to distinct *y positions* (and every elements of a same rank have the same y position).

FMs for visualization designs. The second step aims at defining, with the support of the reference mapping FM, a FM specific to each visualization designs that the visualization domain engineer wants to integrate. For example, the FM specific to the classical tree designs is represented in Figure 7. They describe the different visual variables contained in the visualization and their associated datatypes following the FM for visual variables. Features marked as optional correspond to visual variables that can be used in such visualizations but are not mandatory. For example, a treemap can be colored but it's not a requirement. On the contrary, to be a meaningful choice, the size of treemap elements must carry an information, thus the size is required for this visualization.

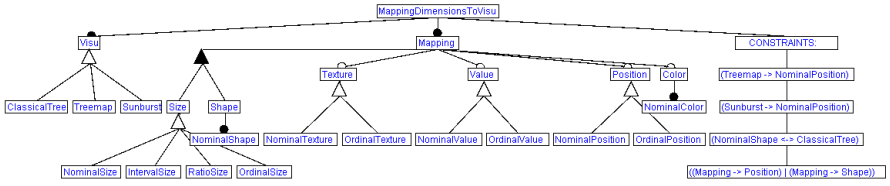

Fig. 10 Visualization Mapping FM (exported from FAMILIAR)

FM for Visualization. The third step is a merge operation of all the FMs specific to each visualization design. This operation is managed by the *Familiar* tool [1] and allows to obtain a single FM that integrates all the variability of the involved FMs. The use of Familiar in this case is particularly interesting as it provides a

straightforward solution to propose different FMs for different visualization subsets. The Figure 10 illustrates the result of the merge of the treemap FM, the classical tree FM and the sunburst FM, as depicted by Familiar. Some constraints allowing to manage the variability of the visual variables depending on the type of visualization designs have been generated by the tool. The merge resulting FM is the visualization FM that the designer has to configure to select the adapted visualization design.

3.2 Visualization application engineering

Once the visualization domain engineering is complete, the visualization application engineering phase consists mainly in the configuration of the different feature models. The phase is composed of the following steps:

1. The configuration visualization engineer will decide which subset from the original dataset must be present in the resulting visualization.
2. The visualization FM is *automatically* restricted thanks to extra-constraints to reduce the configuration space. Consequently, it allows only visualizations that are relevant for the selected data subset.
3. The configuration visualization engineer configures the restricted visualization FM to choose the visualization and its details among those that are still available.
4. The configuration visualization engineer maps the selected data to the adequate visual variables.

3.2.1 Data-subset configuration

The configuration visualization engineer configures the dataset FM defined during the visualization domain engineering phase (see Section 3.1.1) depending on the dimensions and measures involved in the specific configuration user task considered. The result is a subset of the data with their corresponding datatypes.

3.2.2 Visualization domain restriction

This phase is performed automatically by the system after the data subset configuration. The configuration of the dataset FM leads to establish a combination of different datatypes to visualize. Resultantly, it is necessary to generate constraints on the visualization FM about the datatypes that are mandatory following the configuration of the dataset FM.

The generation of the additional constraints is based on the subset of datatypes contained in the valid configuration of the dataset FM. The subset of datatypes is defined as follows:

Definition 1 (Datatypes subset). Given a configuration $\mathscr{C}$, the datatypes subset $\mathscr{D}_{\mathscr{C}}$ of this configuration is the tuple $\{n,o,i,r\}$ (with $n,o,i,r \in \mathbb{N}$) where:

- n is the number of features in $\mathscr{C}$ prefixed by `Nominal`;
- o is the number of features in $\mathscr{C}$ prefixed by `Ordinal`;
- i is the number of features in $\mathscr{C}$ prefixed by `Interval`;
- r is the number of features in $\mathscr{C}$ prefixed by `Ratio`;

Having x features of a given datatype in the data subset implies that the designer must choose x visual variables that support the same datatype in the visualization feature model to obtain a valid configuration. Suppose that the visualization feature model holds y features being of this datatype, the designer must form a combination of x elements out of y (with $y \geq x$) for this datatype. The number of the possible combinations is consequently $\binom{y}{x}$.

Given a visualization feature model V and a datatype d, the set of visual features for this datatype, $\mathscr{V}_{\{V,d\}}$ is the set of features of V that are prefixed by the datatype name d.

Given a set of features F we define $C(F,n)$ as the disjunction of all the conjunctions of n distinct elements of F.

With these definitions, the set of generated constraints is defined as follow:

Definition 2 (Datatypes subset constraints).

Given a datatype subset $\mathscr{D}_{\mathscr{C}} = \{n,o,i,r\}$ and a visualization feature model V, the datatypes subset constraints of $\mathscr{D}_{\mathscr{C}}$ and V is the following:

$\{C(\mathscr{V}_{\{V,\texttt{Nominal}\}},n), C(\mathscr{V}_{\{V,\texttt{Ordinal}\}},o), C(\mathscr{V}_{\{V,\texttt{Interval}\}},i),$
$C(\mathscr{V}_{\{V,\texttt{Ratio}\}},r)\}$

As example, let `NominalViz` the set of features representing a `Nominal` datatype in the visualization FM:
$\texttt{NominalViz} = (\texttt{NominalTexture}, \texttt{NominalColor},$
$\texttt{NominalShape})$

If the configuration visualization engineer chooses 2 `Nominal` datatypes in the dataset FM, he has to choose exactly 2 features among `NominalViz`. The number of possible combinations is $\binom{3}{2}$. The constraint can be expressed as follows:
$(\texttt{NominalTexture} \wedge \texttt{NominalColor} \wedge \neg\texttt{NominalShape})$
$\vee(\texttt{NominalTexture} \wedge \texttt{NominalShape} \wedge \neg\texttt{NominalColor})$
$\vee(\texttt{NominalColor} \wedge \texttt{NominalShape} \wedge \neg\texttt{NominalTexture})$

Once the data subset constraints are generated, they are inserted into the visualization FM with Familiar and the configuration visualization engineer can then configure the visualization FM.

For a given datatype, the configuration visualization engineer can select in the dataset FM a number of features representing this datatype greater than the number of existing visual variables expressing this datatype in the visualization FM. In

a such situation, none of the proposed visualizations would fit his needs. Thus, either the configuration visualization engineer needs to narrow the data subset or the visualization domain engineer needs to extend the visualization catalog with new visualization designs in order to obtain a visualization FM with valid configurations for the generated constraints.

3.2.3 Visualization configuration

The constraints induced by the configuration of the data feature model and the constraints resulting from the merge of the feature models specific to each visualization designs (see Figure 10) restrict the possible configurations of the visualization FM. The configuration visualization engineer chooses his visualization design among the visualization designs which are still available after this restriction.

3.2.4 Data mapping

The configuration of the visualization FM allows to determine the visualization design, its visual variables and the datatypes that can be expressed by these visual variables. We can define a matrix for each datatype involved: along one axis, we find the candidate visual variables for this datatype, and along the other axis, the data of this type that were selected during the configuration of the dataset FM. The configuration visualization engineer has to designate for each matrix how the data are mapped to the visual variables.

To support this task, the configuration engineer can rely on the length of the visual variables. Each visual variable has a specific *length*, which is the number of distinct values that it can take and that can be effectively perceived by the human's perceptual system. The length of each variable is given in Figure 8. The designer has to ensure that the length of the selected variable is equal or larger than the number of elements of its mapped datatype. Stated another way, the visual variables must at least be able to depict all the elements of the datatype. By applying this rule, the number of candidate variables can be reduced.

Another factor that can reduce the set of candidate variables is the choice of variables that are deliberately not bind to any selected data from the data feature model. This choice is supported by the fact that the configuration engineer wants to keep free some of the visual variables to allow extra information to be added later on the diagram. Indeed, every variable that is mapped to a datatype cannot be easily reused to convey a new meaning on the same diagram. The set of free variables forms the secondary notation, while the primary notation is defined by the set of bound visual variables. For instance, if we consider that it would be important to be able to point out one or several specific elements on a diagram, the variable *color* should be kept free. Color is a variable that is easily and almost instantly perceived. Moreover, the human's perceptual system is able to isolate all occurrences of a given color on a diagram, and hence the color is really appropriate for pointing out elements.

4 Application

In this section we apply our approach on the GPL product line introduced in section 2. In particular, we want to select an appropriate visualization for the following task: "understanding the consequences of feature selection".

4.1 Visualization domain engineering

The visualization domain engineering phase consists in the definition of the dataset FM and the visualization FM. The dataset FM and the visualization FM used for this application case are those illustrated in Figure 4 and in Figure 10. The visualization FM allows to configure a classical tree visualization design, a treemap or a sunburst but can be extended to accommodate new visualizations.

4.2 Visualization application engineering

The same steps as those described in section 3.2 have to be followed in order to complete the visualization application engineering phase for our example.

4.2.1 Data-subset configuration

This step consists in selecting only the features on the dataset FM which are useful for the analysis of the consequences of a feature choice on the other features. Understanding the consequences of a choice implies to visualize the FM and to be able to detect the feature states that become selected or rejected following this choice.

We describe the features selected on the dataset FM as dimensions, then the features selected as measures and finally, the features selected as derived measures.

Dimensions. As we need to visualize the FM, the `Features` dimension is necessary. Consequently the set of following features are selected:

{`DataFM`, `Dimensions`, `Features`, `HierarchyF`,
`LinkF`, `NominalLinkF`, `RankF`, `OrdinalRankF`}

Measures. In order to know the state of the features after the choice, the measure `FeatureState` has to be selected on the dataset FM. Consequently, the following features are selected:

{`Measures`, `FeatureState`, `NominalFS`}

The derived measures. Finally, to be able to detect a feature state that become selected or rejected, the derived measure `FeatureStateChange` is needed. The following features are thus selected:

{DerivedMeasures, FeatureStateChange, NominalFSC}

4.2.2 Visualization domain restriction

The configuration of the dataset FM leads to the generation of some constraints on the visualization FM in accordance with the explanations given in 3.2.2. They imply that the final configuration of the visualization FM contains exactly three visual variables representing a nominal datatype and one representing an ordinal datatype.

4.2.3 Visualization configuration

The visualization domain restriction does not restrict the choices of global visualization designs in the visualization FM. Hence, in Figure 10, the three children of the feature Visu (ClassicalTree, Treemap, Sunburst) can be selected. We select the ClassicalTree feature in order to compare the design resulting from our methodology with the classical tree visualization of the existing configuration tools. As explained in Section 3.1.2, in the classical tree design, the relationship between the parent and the children is usually mapped to the shape as nominal. The rank is usually mapped to the Position visual variable as ordinal. Resultantly, we select these visual variables in order to map them to the features LinkF and RankF of the dataset FM. The constraints generated in 4.2.2 require three visual variables to be able to express the nominal datatypes. Given that Shape and Position are already selected, we need to select two other visual variables among Texture, Color, Value and Size. We choose Texture and Color. The constraints require also one visual variable to depict the datatype ordinal. This constraint is already resolved due to the previous selection of the Position visual variable as ordinal. Finally, we obtain the following configuration:

{MappingDimensionsToVisu, Visu, ClassicalTree,
Mapping, Shape, NominalShape,
Texture, NominalTexture,
Position, OrdinalPosition,
Color, NominalColor}

4.2.4 Data mapping

As described in Section 3.2.4, for each datatype existing in the visualization configuration, we define a matrix with, along one axis, the possible visual variables for the datatype concerned and along the other axis, the set of features of this type that were selected during the configuration of the dataset FM.

We decided in 4.2.3 to map the feature LinkF to the Shape visual variable. Consequently, concerning the datatype nominal, two features from the dataset FM have to be mapped to the visual variables Texture and Color: FeatureState

and FeatureStateChange. The matrix obtained is illustrated in Figure 11 and the crosses indicate the resulting mapping. Given that the length of the visual variables Texture and Color (see Figure 8) are both adapted to represent the measures FeatureState and FeatureStateChange, we could choose another mapping.

Concerning the datatype ordinal, we decided in 4.2.3 to map the feature RankF to the Position visual variable. Hence, a matrix definition is irrelevant for the type ordinal.

	Texture	Color
Feature State		×
Feature State Change	×	

Fig. 11 Mapping Nominal Datatype/Visual Variables

4.3 Resulting visualization

The dataset FM configuration, the visualization FM configuration and the data mapping lead us to build a visualization design holding three essential characteristics.

The first characteristic is that the visualization design is a classical tree representing the feature dimension. We choose to represent a feature by a circle with an affixed text label indicating the feature name.

The second characteristic is the color of the node circle circumference that represents the measure FeatureState. We choose the green and the red color to visualize the feature state *selected* and respectively *rejected*. We take a light blue for the feature state *free*.

The third characteristic is the circle circumference line texture that indicates the measure FeatureStateChange. We choose to design a solid line when a feature state becomes selected or rejected after the choice and a dashed line when a feature state was already selected or rejected before the choice.

By following our method with the aim to visualize the impacts of a feature choice, we obtain the visualization design illustrated in Figure 12. The Figure presents the FM configuration view after the selection of the feature StrongC as detailed in 2.

We can compare our result with the configuration views proposed by the configuration tools Feature IDE and S2T2 illustrated in Figures 1 and 2. Whereas on our configuration view, the impacts of the feature StrongC selection are clearly identifiable due to the use of the color and texture on the feature nodes, the impacts are not pointed out on the two other configuration views. S2T2 offers an animation to get a glimpse of the impacts when the stakeholder selects a feature. However, this furtive glimpse is not suitable for the analysis of the selection consequences,

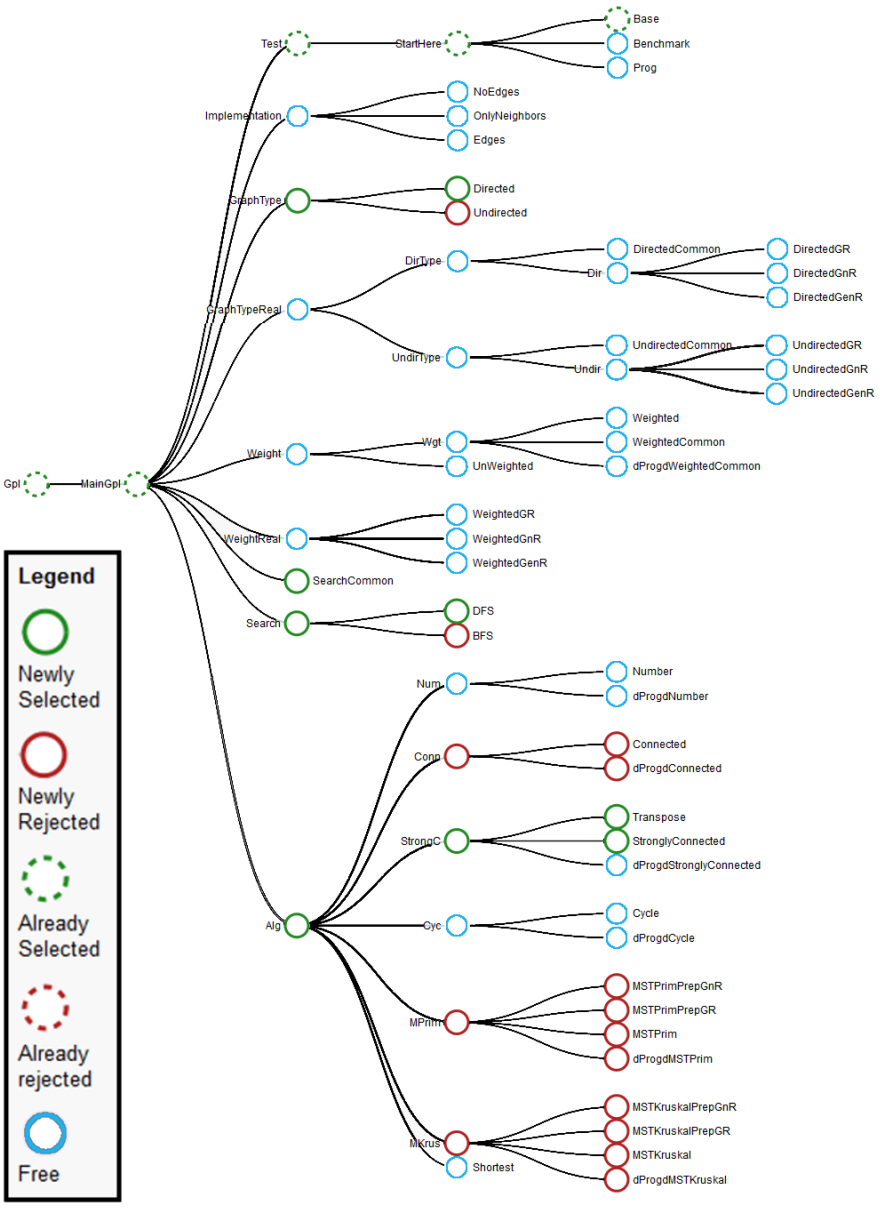

Fig. 12 FM configuration view

particularly if all the consequences are not visible on the same space screen. Feature IDE uses the red and green colors to represent the feature state but does not make the distinction between a selection/rejection of features before/after a given feature choice.

5 Related work

Among the common existing configuration tools, some contain basic FM configuration views [22], others are the result of specific research aiming to integrate more visual support in the configuration tools. They offer additional views [11, 10] or advanced configuration views in order to help the stakeholders to make their choices. S2T2, the tool presented in [12] and illustrated in Figure 2 offers several interesting visualization mechanisms such as, for example, different tree layouts, zoom and pan mechanism, or color usage to indicate the configuration progress. Whilst the designed choices are well documented, there is, to the best of our knowledge, no explicit method to support them. The objective of our method is to (at least partially) automate the reasoning process that leads to such design choices, as shown in the example.

These various visual propositions for the configuration tools reveal a growing interest among SPL researchers to find adapted visualization techniques for the configuration tasks.

For its part, the visualization community has come up with approaches intended to guide the designer towards a solution visualization. In [9], Mackinlay bases his work on the expressiveness and effectiveness of a set of primitive graphical languages. He presents a framework for the development of tools which are able to automatically generate a design for relational information. Whereas the algorithm presented decides on a unique relevant visualization design, our approach leaves the final decision to the configuration visualization engineer, given that several visualizations may satisfy constraints. A methodology based on a connection between the data interpretation aims and the data representation possibilities is presented in [15]. In [24], Zhang gives a classification of relational information displays that can guide the designers to select a visualization solution. More recently, [5] presented a tool that allows to select a visualization design in accordance with the data characteristics and the user's objectives. This study intends to be used in a visual data mining context where the user's objectives are to discover interesting structure inside the data. In contrast, the general aim of a user in your case study is to take the right configuration decision. For that purpose, a clear display of the appropriate data and of their existing intrinsic relationships is needed. Based on these considerations, our approach is suitable for our goal.

6 Conclusions

In this chapter, we presented a SPL approach guiding the visualization engineer to a cognitively optimal visualization design choice for SPL configuration tasks. By following the method described, the engineer is not restricted to a specific visualization solution but can choose among a set of suitable designs for the input data involved. Moreover, the visualization solution thus designed ensures consistency in terms of mapping between the visual variables and the data depicted.

Behind the interest for the engineers of a such approach, the use of feature models to support it deserves to be pointed out. Indeed, this particularity allows an easy extensibility of the basic principles we laid down. Moreover, by adapting the dataset FM, our approach can be useful for application domains other than the SPL configuration tasks. Finally, the fact that FMs can be encoded as formal decision models opens the possibility to partial automation of the approach as applied with FAMILIAR in this paper.

Future work will consist in improving the method according to three different aspects. The first concerns the dataset FM. An extension of its definition is necessary in order to take into account the configuration tasks carried out on FMs with a greater expressiveness (i.e. attributes, cardinalities). Moreover, we would like to integrate some quantitative metrics such as the size of the FM and manage the consequences on the resulting visualizations.

The second aspect relates on the visualization solutions proposed. At the current step of our approach, the design depends only on the data involved in the considered user task. As explained in [24], we need to take into account the user task characteristics to improve our propositions of solutions. Furthermore, the dynamic parts of the visualization solutions were deliberately set aside.

The third aspect concerns the improvement of the approach itself by adding guidance to the decisions left at the moment to the visualization engineer. Additionally, we would like to validate our approach by performing an empirical study as described in [17].

7 Acknowledgements

This work was partly supported by the European Commission (FEDER IDEES/CO-INNOVATION).

References

1. Acher, M., Collet, P., Lahire, P., France, R.B.: Familiar: A domain-specific language for large scale management of feature models. Science of Computer Programming **78**(6), 657–681 (2013)
2. Bertin, J.: Semiology of graphics: Diagrams, networks, maps (wj berg, trans.). Madison, WI: The University of Wisconsin Press, Ltd (1983)
3. Cleveland, W.C., McGill, M.E.: Dynamic graphics for statistics. CRC Press, Inc. (1988)
4. Deelstra, S., Sinnema, M., Bosch, J.: A product derivation framework for software product families. In: Software Product-Family Engineering, pp. 473–484. Springer (2004)
5. Guettala, A.E.T., Bouali, F., Guinot, C., Venturini, G.: A user assistant for the selection and parameterization of the visualizations in visual data mining. In: Information Visualisation (IV), 2012 16th International Conference on, pp. 252–257. IEEE (2012)
6. Johnson, B., Shneiderman, B.: Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In: Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on, pp. 284–291. IEEE (1991)

7. Lopez-Herrejon, R.E., Batory, D.: A standard problem for evaluating product-line methodologies. In: Generative and Component-Based Software Engineering, pp. 10–24. Springer (2001)
8. MacEachren, A.M.: How maps work: representation, visualization, and design. Guilford Press (2004)
9. Mackinlay, J.: Automating the design of graphical presentations of relational information. Acm Transactions On Graphics (Tog) **5**(2), 110–141 (1986)
10. Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K.: Visualization and exploration of optimal variants in product line engineering. In: Proceedings of the 17th International Software Product Line Conference, pp. 111–115. ACM (2013)
11. Nöhrer, A., Egyed, A.: C2o: a tool for guided decision-making. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, pp. 363–364. ACM (2010)
12. Pleuss, A., Botterweck, G.: Visualization of variability and configuration options. International Journal on Software Tools for Technology Transfer **14**(5), 497–510 (2012)
13. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin (2005). DOI 10.1007/3-540-28901-1
14. Ravat, F., Teste, O., Zurfluh, G.: Algebre olap et langage graphique. arXiv preprint arXiv:1005.0213 (2010)
15. Robertson, P.K.: A methodology for choosing data representations. IEEE Computer Graphics and Applications **11**(3), 56–67 (1991)
16. Schulz, H.J.: Treevis. net: A tree visualization reference. Computer Graphics and Applications, IEEE **31**(6), 11–15 (2011)
17. Sedlmair, M., Meyer, M., Munzner, T.: Design study methodology: Reflections from the trenches and the stacks. Visualization and Computer Graphics, IEEE Transactions on **18**(12), 2431–2440 (2012)
18. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: The variability model of the linux kernel. VaMoS **10**, 45–51 (2010)
19. Stasko, J., Catrambone, R., Guzdial, M., McDonald, K.: An evaluation of space-filling information visualizations for depicting hierarchical structures. International Journal of Human-Computer Studies **53**(5), 663–694 (2000)
20. Steger, M., Tischer, C., Boss, B., Müller, A., Pertler, O., Stolz, W., Ferber, S.: Introducing pla at bosch gasoline systems: Experiences and practices. In: Software Product Lines, pp. 34–50. Springer (2004)
21. Stevens, S.S.: On the theory of scales of measurement (1946)
22. Thum, T., Kstner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming (2012)
23. Wilkinson, L.: The grammar of graphics. Springer (2012)
24. Zhang, J.: A representational analysis of relational information displays. International Journal of Human-Computer Studies **45**(1), 59–74 (1996)